IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Deboer et al.**	§	
	§	Group Art Unit: **2443**
Serial No.: **10/718,297**	§	
	§	Examiner: **Sikri, Anish**
Filed: **November 20, 2003**	§	
	§	Confirmation No.: **9784**
For: **Extensible Mechanism for Executing Server Side Code**	§	
	§	

35525

PATENT TRADEMARK OFFICE
CUSTOMER NUMBER

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF (37 C.F.R. 41.37)

This brief is in furtherance of the Notice of Appeal, filed in this case on November 23, 2009.

A fee of $540.00 is required for filing an Appeal Brief. Please charge this fee to IBM Corporation Deposit Account No. 09-0447. No additional fees are believed to be necessary. If, however, any additional fees are required, I authorize the Commissioner to charge these fees which may be required to IBM Corporation Deposit Account No. 09-0447. No extension of time is believed to be necessary. If, however, an extension of time is required, the extension is requested, and I authorize the Commissioner to charge any fees for this extension to IBM Corporation Deposit Account No. 09-0447.

REAL PARTY IN INTEREST

The real party in interest in this appeal is the following party: International Business Machines Corporation of Armonk, New York.

RELATED APPEALS AND INTERFERENCES

This appeal has no related proceedings or interferences.

STATUS OF CLAIMS

A. TOTAL NUMBER OF CLAIMS IN APPLICATION

The claims in the application are: 1-35

B. STATUS OF ALL THE CLAIMS IN APPLICATION

Claims canceled: 34

Claims withdrawn from consideration but not canceled: None

Claims pending: 1-33 and 35

Claims allowed: None

Claims rejected: 1-33 and 35

Claims objected to: None

C. CLAIMS ON APPEAL

The claims on appeal are: 1-33 and 35

STATUS OF AMENDMENTS

No amendment was filed after final rejection.

SUMMARY OF CLAIMED SUBJECT MATTER

A. CLAIM 1 - INDEPENDENT

The subject matter of claim 1 is directed to a method of executing code in a client-server environment by defining an extensible mechanism for executing said code on a server that, when deployed on a computer system, adapts an integrated development environment (IDE) to handle new code types as described in the specification in paragraphs [0004], [0032], [0034]-[0036] and Figure 3, 308, Figure 4, 308. Claim 1 recites identifying an input object on a client system, the input object identifying code for executing on one of a plurality of servers as described in the specification in paragraphs [0042], [0048] and Figure 6, 604. Claim 1 recites processing the input object to identify the code for executing on the one of the plurality of servers as described in the specification in paragraphs [0035], [0048] and Figure 6, 608. Claim 1 recites generating, in response to identifying the code for executing on the one of the plurality of servers, code for accessing the code for executing on the one of the plurality of servers as described in the specification in paragraphs [0035], [0042], [0048] and Figure 6, 610. Claim 1 recites processing the generated code to determine the one of the plurality of servers on which to execute the code, each code for executing on a server being able to execute on a particular server as described in the specification in paragraphs [0035], [0043], [0048]-[0049] and Figure 6, 614. Claim 1 recites enabling the determined server to access the code for executing on the one of the plurality of servers as described in the specification in paragraphs [0035], [0043], [0050] and Figure 6, 616, 618. Claim 1 recites identifying, based on the accessed code for executing on the one of the plurality of servers, one of a plurality of client applications for allowing the determined server to interact with the client system during processing of the code for executing on the one of the plurality of servers as described in the specification in paragraphs [0035], [0045], [0051] and Figure 6, 622, 624. Claim 1 recites processing the code for executing on the one of the plurality of servers as described in the specification in paragraphs [0035], [0043], [0050].

B. CLAIM 2 – DEPENDENT

The subject matter of claim 2 is directed to the method of claim 1 wherein processing the input object to identify the code for executing on the one of the plurality of servers includes using

a view list of at least one input element for processing a type of code identified by the input object, processing the generated code includes using a server list of at least one server element for determining the one of the plurality of servers, and identifying the one of the plurality of client applications includes using a launcher list of at least one client element for launching the one of the plurality of client applications as described in the specification in paragraph [0005] and Figure 3, 310.

C. CLAIM 3 – DEPENDENT

The subject matter of claim 3 is directed to the method of claim 2 wherein at least one of the view list, server list and launcher list is extensible to accommodate additional respective elements as described in the specification in paragraph [0006] and Figure 3, 310.

D. CLAIM 4 – DEPENDENT

The subject matter of claim 4 is directed to the method of claim 2 comprising maintaining at least one of the view list, server list and launcher list as described in the specification in paragraphs [0005], [0014], [0032] and Figure 3, 310.

E. CLAIM 5 – DEPENDENT

The subject matter of claim 5 is directed to the method of claim 4 wherein the step of maintaining at least one of the view list, server list and launcher list comprises extending any of the view list, server list and launcher list as described in the specification in paragraphs [0005], [0014] and Figure 3, 310.

F. CLAIM 6 – DEPENDENT

The subject matter of claim 6 is directed to the method of claim 1 wherein processing the input object comprises analyzing the input object to determine an input object element for processing the input object and processing the input object using the determined input object element as described in the specification in paragraphs [0007], [0032], [0048] and Figure 6, 608.

G. CLAIM 7 – DEPENDENT

The subject matter of claim 7 is directed to the method of claim 6 including processing user input to determine the input object element as described in the specification in paragraphs [0007], [0032], [0042], [0048] and Figure 6, 608.

H. CLAIM 8 – DEPENDENT

The subject matter of claim 8 is directed to the method of claim 1 wherein processing the generated code comprises analyzing a server element for enabling a deployable object and processing the deployable object using the determined server element as described in the specification in paragraphs [0035], [0043], [0048]-[0049] and Figure 6, 614.

I. CLAIM 9 – DEPENDENT

The subject matter of claim 9 is directed to the method of claim 8 including processing user input to determine the server element as described in the specification in paragraphs [0035], [0042]-[0043], [0048]-[0049] and Figure 6, 614.

J. CLAIM 10 – DEPENDENT

The subject matter of claim 10 is directed to the method of claim 1 wherein identifying the one of the plurality of client applications comprises analyzing a launchable object to determine a client element for processing the launchable object and processing the launchable object using the determined client element as described in the specification in paragraphs [0035], [0045], [0051] and Figure 6, 622, 624.

K. CLAIM 11 – DEPENDENT

The subject matter of claim 11 is directed to the method of claim 10 including processing user input to determine the server element as described in the specification in paragraphs [0035], [0042], [0045], [0051] and Figure 6, 622, 624.

L. CLAIM 12 – INDEPENDENT

The subject matter of claim 12 is directed to an extensible mechanism for executing server side code in a client-server environment that, when deployed on a computer system, adapts an integrated development environment (IDE) to handle new code types as described in the specification in paragraphs [0004], [0032], [0034]-[0036] and Figure 3, 308, Figure 4, 308. Claim 12 recites a view mechanism for processing an input object identifying code for executing on one of a plurality of servers and outputting a deployable object as described in the specification in paragraphs [0008], [0032], [0040], [0042], and Figure 3, 310, Figure 5, 502. Claim 12 recites a server mechanism for processing the deployable object to determine the one of the plurality of servers for executing the code and to enable the deployable object to execute on the one of the plurality of servers, said second mechanism outputting a launchable object as described in the specification in paragraphs [0008], [0032], [0040], [0043]-[0044], and Figure 3, 310, Figure 5, 504. Claim 12 recites a launcher mechanism for processing the launchable object to determine one of a plurality of client applications for launching the code on the one of the plurality of servers as described in the specification in paragraphs [0008], [0032], [0040], [0045], and Figure 3, 310, Figure 5, 506.

M. CLAIM 13 – DEPENDENT

The subject matter of claim 13 is directed to the extensible mechanism of claim 12 wherein said view mechanism comprises a view list of at least one input object element, each input object element processing a type of code identified by the input object for outputting the deployable object as described in the specification in paragraphs [0005], [0042], [0049] and Figure 3, 310, Figure 5, 502.

N. CLAIM 14 – DEPENDENT

The subject matter of claim 14 is directed to the extensible mechanism of claim 13 wherein said view list is extensible to accommodate additional respective elements as described in the specification in paragraph [0006] and Figure 3, 310.

O. CLAIM 15 – DEPENDENT

The subject matter of claim 15 is directed to the extensible mechanism of claim 12 wherein said server mechanism comprises a server list of at least one server element, each server element enabling the deployable object to execute on a particular server and processing the deployable object for outputting a launchable object as described in the specification in paragraphs [0005], [0043]-[0044] and Figure 3, 310, Figure 5, 504.

P. CLAIM 16 – DEPENDENT

The subject matter of claim 16 is directed to the extensible mechanism of claim 15 wherein said server list is extensible to accommodate additional respective elements as described in the specification in paragraph [0006] and Figure 3, 310.

Q. CLAIM 17 – DEPENDENT

The subject matter of claim 17 is directed to the extensible mechanism of claim 12 wherein said launcher mechanism comprises a launcher list of at least one client element, each client element enabling the launchable object to execute on a particular client for launching the code on the particular server as described in the specification in paragraphs [0005], [0045] and Figure 3, 310, Figure 5, 506.

R. CLAIM 18 – DEPENDENT

The subject matter of claim 18 is directed to the extensible mechanism of claim 17 wherein said launcher list is extensible to accommodate additional respective elements as described in the specification in paragraph [0006] and Figure 3, 310.

S. CLAIM 29 – INDEPENDENT

The subject matter of claim 29 is directed to a computer readable media storing instructions to be executed by a processor of a computer system, said processor of the computer system executing an integrated development environment (IDE) for generating code for executing in a client-server environment, said instructions defining an extensible mechanism for executing said code on a server that, when deployed on said computer system, adapts said IDE for handling new

code types as described in the specification in paragraphs [0004], [0032], [0034]-[0036] and Figure 3, 308, Figure 4, 308. Claim 29 recites processing an input object identifying code for executing on one of a plurality of servers, said processing using a view list of at least one input object element, each input object element processing a type of code identified by the input object to output a deployable object as described in the specification in paragraphs [0035], [0048] and Figure 6, 608. Claim 29 recites processing the deployable object using a server list of at least one server element to determine the one of the plurality of servers for executing the code, each server element enabling the deployable object to execute on a particular server and outputting a launchable object as described in the specification in paragraphs [0035], [0043], [0048]-[0049] and Figure 6, 614. Claim 29 recites processing the launchable object using a launcher list of at least one client element to determine a client for launching the code on the one of the plurality of servers as described in the specification in paragraphs [0035], [0045], [0051] and Figure 6, 622, 624.

T. CLAIM 32 – INDEPENDENT

The subject matter of claim 32 is directed to a method of maintaining an extensible mechanism for executing server side code in a client-server environment by defining an extensible mechanism for executing said code on a server that, when deployed on a computer system, adapts and integrated development environment (IDE) to handle new code types as described in the specification in paragraphs [0004], [0032], [0034]-[0036] and Figure 3, 308, Figure 4, 308. Claim 32 recites maintaining at least one of a view list of at least one input object element, each input object element processing a type of code identified by the input object to output a deployable object as described in the specification in paragraphs [0005], [0042], [0049] and Figure 3, 310, Figure 5, 502. Claim 32 recites maintaining a server list of at least one server element to determine one of a plurality of servers for executing the code, each server element enabling the deployable object to execute on a particular server and outputting a launchable object as described in the specification in paragraphs [0005], [0043]-[0044] and Figure 3, 310, Figure 5, 504, Claim 32 recites maintaining a launcher list of at least one client element to determine one of a plurality of client applications for launching the code on the one of the plurality of servers as described in the specification in paragraphs [0005], [0045] and Figure 3, 310, Figure 5, 506.

U. CLAIM 35 – DEPENDENT

The subject matter of claim 35 is directed to the computer readable media of claim 29, further comprising performing a compatibility test of the input object, deployable object, and launchable object prior to processing the input object, and displaying a result of the compatibility test to a user as described in the specification in paragraph [0052].

GROUND OF REJECTION TO BE REVIEWED ON APPEAL

The ground of rejection to review on appeal is as follows:

A. GROUND OF REJECTION 1

The rejection of claims 1-33 and 35 under 35 U.S.C. § 103 as being unpatentable over Christfort et al. (U.S. Publication No. 2002/0078168) in view of Uszok et al. (U.S. Publication No. 2004/0205772).

ARGUMENT

A. GROUND OF REJECTION 1 (Claims 1-33 and 35)

The Examiner has rejected claims 1-33 and 35 under 35 U.S.C. § 103 as being unpatentable over Christfort et al. (U.S. Publication No. 2002/0078168), hereinafter "*Christfort*" in view of Uszok et al. (U.S. Publication No. 2004/0205772), hereinafter "*Uszok*".

The Examiner states:

> Consider Claim 29, Christfort disclosed A computer readable media storing instructions to be executed by a processor of a computer system (Christfort, [0298]), said processor of the computer system executing an integrated development environment (IDE) for generating code for executing in a client-server environment (Christfort, [0080], Christfort discloses an IDE for executing code), to: process an input object identifying code for executing on one of a plurality of servers (Christfort, [0022], Christfort disclosed on identifying several types of input objects which can be used for coding for the application) said processing using a view list of at least one input object element (Christfort, [0022], Christfort discloses on view several input object codes which are presented to the user), each input object element processing a type of code identified by the input object to output a deployable object (Christfort, [0022], Christfort discloses on how the application code is selected by an object on the interface, and the application code may be executed in a response to a request for a service from an end user); process the deployable object using a server list of at least one server element to determine the one of the plurality of servers for executing the code (Christfort, [0062], Christfort does show the server list for example containing host servers on a portal page), each server element enabling the deployable object to execute on a particular server and outputting a launchable object (Christfort, [0094]-[0095], Christfort disclosed on how objects/created applications are launched via the system); and process the launchable object using a launcher list of at least one client element to determine a client for launching the code on the one of the plurality of servers (Christford, [0093], Christfort indicates on how portal-to-go XML document or application program containing the code generates the output, and how the output is launched by the system).
>
> But Christford does not explicitly disclose said instructions defining an extensible mechanism for executing said code on a server that, when deployed on said computer system, adapts said IDE for handling new code types
>
> Nonetheless, Uszok discloses said instructions defining an extensible mechanism for executing said code on a server that (Uszok, [0129], Uszok, discloses how the SDK can be used to generate a bot code "executable code" mechanism), when deployed on said computer system, adapts said IDE for handling new code types (Uszok, [0129], Uszok discloses on how the bot code is able to implement a selection of predetermined protocols set in the SDK).

> Both Chirstford, and Uszok provide features related to SDK/IDE management. Therefore one of ordinary skill in the art would have been motivated to combine the teachings since both are within the same environment.
>
> Therefore, it would have been obvious to a person skilled in the art at the time of the invention was made to incorporate executable code mechanism by the SDK/IDE, taught by Uszok, in the system of Christford for the purpose of efficient code execution mechanism.

Final Office Action dated August 21, 2009, pages 3-5.

The Examiner bears the burden of establishing a *prima facie* case of obviousness based on prior art when rejecting claims under 35 U.S.C. § 103. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992). The prior art reference (or references when combined) must teach or suggest all the claim limitations. *In re Royka,* 490 F.2d 981, 180 USPQ 580 (CCPA 1974). In determining obviousness, the scope and content of the prior art are... determined; differences between the prior art and the claims at issue are... ascertained; and the level of ordinary skill in the pertinent art resolved. Against this background the obviousness or non-obviousness of the subject matter is determined. *Graham v. John Deere Co.*, 383 U.S. 1 (1966). "Often, it will be necessary for a court to look to interrelated teachings of multiple patents; the effects of demands known to the design community or present in the marketplace; and the background knowledge possessed by a person having ordinary skill in the art, all in order to determine whether there was an apparent reason to combine the known elements in the fashion claimed by the patent at issue." *KSR Int'l. Co. v. Teleflex, Inc.*, No. 04-1350 (U.S. Apr. 30, 2007). "*Rejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness.* Id. (citing *In re Kahn*, 441 F.3d 977, 988 (CA Fed. 2006))."

Independent claim 29, which is representative of independent claims 1, 12, and 32 with regard to similarly recited subject matter, reads as follows:

> 29. A computer readable media storing instructions to be executed by a processor of a computer system, said processor of the computer system executing an integrated development environment (IDE) for generating code for executing in a client-server environment, said instructions defining an extensible mechanism for executing said code on a server that, when deployed on said computer system, adapts said IDE for handling new code types by:

processing an input object identifying code for executing on one of a plurality of servers, said processing using a view list of at least one input object element, each input object element processing a type of code identified by the input object to output a deployable object;

processing the deployable object using a server list of at least one server element to determine the one of the plurality of servers for executing the code, each server element enabling the deployable object to execute on a particular server and outputting a launchable object; and

processing the launchable object using a launcher list of at least one client element to determine a client for launching the code on the one of the plurality of servers.

The combination of *Christfort* and *Uszok* does not teach the feature of instructions defining an extensible mechanism for executing said code on a server that, when deployed on the computer system, adapts the integrated development environment for handling new code types. Appellants agree with the Examiner's statement that *Christfort* does not teach this feature (Final Office Action dated August 21, 2009, page 4). Instead, the Examiner relies on the following section of *Uszok* as teaching this feature:

> **[0129]** A bot developer kit (SDK) can be arranged to generate bot code to implement any of a selection of predetermined protocols. For example, a set of business function protocols can be built into the SDK to support rapid coding of bots to carry out those business functions. Through this implementation of protocols and standard data schema, interoperability of bots is enabled over a wide variety of practical applications. A Bot platform (botServer/ botExecutor) participates in protocol execution in following ways: First, it verifies the protocol compatibility when a bot accesses the appropriate meeting place. Next, the platform traces the bots' and plug-ins' states and documents passed between them. It excludes from interaction those bots that do not conform to the agreed protocol.

Uszok, paragraph [0129].

The cited section of *Uszok* above discloses that a bot developer kit (SDK) may be arranged to generate bot code to implement any of a selection of predetermined protocols. An example is given in which a set of business function protocols can be built into the SDK to support rapid coding of bots to carry out those business functions.

However, *Uszok* does not teach defining an *extensible mechanism* that *adapts the integrated development environment to handle new code types*. *Uszok* teaches generating bot

code using a software development kit to implement predetermined protocols. In contrast, claim 29 of the present invention recites a process for modifying a software development kit itself using the extensible mechanism, which allows adapting to new code types. The presently claimed invention enables the IDE to adapt to new code types prior to installation on the server that is identified in order for the server to be able to run the new code. *Uszok* does not mention anything about having an IDE or SDK "adapt" to the new instructions using an extensible mechanism. No such modification of the IDE/SDK itself is performed in *Uszok*. Consequently, *Christfort* and *Uszok* do not teach instructions defining an extensible mechanism for executing said code on a server that, when deployed on the computer system, adapts the integrated development environment to handle new code types as recited in the presently claimed invention.

The combination of *Christfort* and *Uszok* also does not teach the feature of processing an input object identifying code for executing on one of a plurality of servers, said processing using a view list of at least one input object element, each input object element processing a type of code identified by the input object to output a deployable object. The Examiner points to the following section of *Christfort* as teaching this feature:

> **[0022]** Techniques are provided for developing applications online. According to one aspect, applications for services are developed online. A request is received from a user running a browser on a client to develop a new application. In response to the request, a first electronic document, such as a web page, is provided to the client for display on the user's browser of an application development interface. The interface may include several types of objects, including an edit field for typing in code for an application. The user submits the application code by selecting an object on the interface, such as a submit button. The application code is stored at a server that is remote from the client and from where the application code may be executed in response to a request for service from an end user. Similarly, the interface may be used to retrieve and edit existing application code, or to test, deploy, or delete existing application code.

Christfort, paragraph [0022].

Paragraph [0022] of *Christfort* discloses displaying a web page to a user on the browser of an application development interface. The interface includes several input objects that allow the user to type in code for an application. Application code is submitted by selection of an object on the interface, and the application code may be executed in a response to a request for a

service from an end user.

However, *Christfort* is directed to the creation and modification of on-line applications using an on-line software development kit. *Christfort* only teaches receiving input code in an edit field of an interface and receiving a selection of an object (e.g., submit button) on the interface to receive submission of the input code from the user. In contrast, the presently claimed invention recites processing an input object using a view list. *Christfort* makes no mention of a view list for processing input objects. The presently claimed invention also recites an input object that identifies code for execution on one of a plurality of servers. Paragraph [0022] does not mention anything about processing an input object for the purpose of subsequently identifying code to be executed by a server. *Christfort* only teaches receiving input code in an edit field of an interface. *Christfort* further fails to teach a view list comprising at least one input object element, or that an input object element in the view list processes a type of code identified by the input object to output a deployable object. *Christfort* merely teaches receiving input code in an edit field of an interface, receiving a selection of an object on the interface to receive submission of the input code, and execution of an application built by the SDK subsequent to a service request. *Christfort* makes no mention of a view list of input object elements that process code identified by the input object, or that processing by such an input object element in the view list outputs a deployable object.

The combination of *Christfort* and *Uszok* further does not teach the feature of processing the deployable object using a server list of at least one server element to determine the one of the plurality of servers for executing the code, each server element enabling the deployable object to execute on a particular server and outputting a launchable object. The Examiner points to the following sections of *Christfort* as teaching these features:

> **[0062]** Host server 110 may be implemented on one or more servers at an intermediary, such as a hosting service provider, also known as a host provider or simply as a host. The function of the host is to install and maintain applications, such as on host server 110, that are developed by either the host provider or other application developers. The applications are typically part of a service, such as a web site, a paging service, or a telecommunications service. The host may also provide "partial" or "shared" hosting of applications in which the applications are stored on servers associated with the application developer or service provider, but the applications may be accessed via the host. Partial or shared hosting of applications is distinguished from portal applications that are stored on servers

> associated with the application developer or service provider but which are not accessed via the host. End users access the services offered by other parties and companies via the host by interacting with the hosted and partially hosted applications.

Christfort, paragraph [0062].

> **[0094]** As soon as a service has been created and/or revised, end users or customers that can connect to the network on which the server resides (e.g., the Internet) can access the service. The process by which the service is accessed may vary based on the type of end user. For example, a desktop computer can connect to the Internet through a dial-up line, a DSL connection, a cable modem, an ISDN connection or many other available methods. WAP phones may connect to the Internet over a wireless connection using a synchronous protocol, such as through a WAP-to- HTTP gateway, or using an asynchronous protocol, such as the simple mail transfer protocol (SMTP) or the short message service (SMS) protocol.
> **[0095]** In general, the end user logs in to the development website and is presented with a list of available services. The end user may select the service that was just created and/or modified. In response to the selection of a service, the application associated with the service is obtained. The application may be associated with a portal-to-go XML document or another document written in a suitable markup language. For example, with hosted applications, the XML may be obtained by retrieving the portal-to-go XML that was saved at a server associated with the intermediary (e.g., the host). For portal applications, a request may be sent to the URL that was specified by the developer for the service. The web server that manages that URL may invoke the portal application and send the portal-to-go XML associated with the portal application back to the server associated with the intermediary. Once the portal-to-go XML for the service is obtained for either a hosted or portal application, the server associated with the intermediary uses the portal-to-go XML to provide the selected service to the client associated with the end user.

Christfort, paragraphs [0094]-[0095].

Paragraph [0062] of *Christfort* discloses a host provider whose function is to install and maintain applications that are part of a service. End users may access the services via the host. However, *Christfort* does not teach processing the deployable object to determine the one of the plurality of servers for executing the code. No processing of a deployable object is disclosed as occurring in *Christfort*, as the target servers in *Christfort* is already known. The process disclosed in *Christfort* goes from a user creating or editing an application directly to saving the

application on a server as a new or existing service, as explicitly disclosed in the following section below:

> **[0091]** When **the user finishes inputting the code for a new application or editing the code for an existing application,** the user can save the code shown in the interface. The "save" code function may be an object, such as a button, included in the interface. By clicking on the object, **the code is submitted, or sent, to a server** remote from the client, such as a server associated with the development provider. If a new application is being created, the user specifies a service name and the code is saved in association with the service name. After saving the new code, the new service name will be displayed in a list of existing services shown when users and end users log into the development website. If an existing application is being modified, the user may choose to save the code under the existing service name or under a new service name.

Christfort, paragraph [0091] (emphasis added).

Thus, while *Christfort* supports a plurality of servers for a hosting environment, *Christfort* does not support the explicitly recited feature of processing the deployable object to determine the one of the plurality of servers for executing the code as recited in the presently claimed invention. The *Christfort* process does not mention a processing step using a deployable object that determines which one of a plurality of servers that will execute the code. Instead, *Christfort* teaches a server side application environment where the user has already chosen the server to target prior to editing and saving the application. Thus, *Christfort* does not disclose a deployable object being processed to determine the one of the plurality of servers for executing the code. In addition, *Christfort* does not disclose a server element that enables the deployable object to execute on a particular server, or that the server element outputs launchable object. *Christfort* also makes no mention of an input object that outputs a deployable object for the type of code, and processing of the deployable object creates an output of a launchable object.

The combination of *Christfort* and *Uszok* also does not teach processing the launchable object using a launcher list of at least one client element to determine a client for launching the code on the one of the plurality of servers. The Examiner points to the following section of *Christfort* as teaching this feature:

> **[0093]** In another embodiment, to create a shared hosted application, the user writes either a portal-to-go XML document or an application program that

> generates a portal-to-go XML document as output. The terms "partially hosted application" or the "shared hosted application" may be used to refer either to the XML document or the application that generates an XML document as output. The shared hosted application may be saved, for example, at the application developer's own website. The user then associates a URL with the shared hosted application using, for example, an HTTP listener/web server that services the application developer's web site. The shared hosted application is then added as a "service" by logging into the development website, or the SDK website, and providing the name of the service and the URL associated with the shared hosted application.

Christfort, paragraph [0093].

Paragraph [0093] discloses how a shared hosted application may be created by a user by writing either to a portal-to-go XML document or application program containing the code to generate a portal-to-go XML document as output, and how the portal-to-go XML document output is launched by the system. However, the presently claimed invention recites processing a launchable object. *Christfort* merely teaches creating server side applications that can run on any type of client. A user in *Christfort* is further required to perform the manual step disclosed in paragraph [0095]:

> **[0095]** In general, the end user logs in to the development website and is presented with a list of available services. The end user may select the service that was just created and/or modified. In response to the selection of a service, the application associated with the service is obtained. The application may be associated with a portal-to-go XML document or another document written in a suitable markup language.

Christfort, paragraph [0095].

A launchable object is defined in paragraph [0041] of the present specification as "an output, such as a Java code object, XML document etc., that contains information on how to access code to be run (i.e. a specific resource) on a particular server, and information about how to route access to the resource, (e.g. firewalls and/or the type of client application that can be used to access the web resource) if the access information is not sufficiently clear. For example, a URL may be constructed and provided to direct access to the resource when a Web browser is intended to consume the object. Thus, while *Christfort* supports any client launching the service

application, *Christfort* makes no mention of pre-configuring a launcher at a client element to support launchable objects. Consequently, *Christfort* does not teach processing the launchable object using a launcher list of at least one client element to determine a client for launching the code on the one of the plurality of servers.

Since claims 2-11, 13-27, 30-31 and 35 depend from claims 1, 12, and 29, respectively, the same distinctions between *Christfort* and *Uszok* and the claimed invention in claims 1, 12, and 29 also apply to these dependent claims. Furthermore, these dependent claims also include additional features not found in the *Christfort* and *Uszok* references.

For example, claim 2 recites processing the input object to identify the code for executing on the one of the plurality of servers includes using a view list of at least one input element for processing a type of code identified by the input object, processing the generated code includes using a server list of at least one server element for determining the one of the plurality of servers, and identifying the one of the plurality of client applications includes using a launcher list of at least one client element for launching the one of the plurality of client applications, and claim 13 recites wherein said view mechanism comprises a view list of at least one input object element, each input object element processing a type of code identified by the input object for outputting the deployable object. While the Examiner refers again to paragraphs [0022], [0062], and [0093] of *Christfort* (reproduced above) as teaching these features, paragraph [0022] merely discloses several types of input objects which can be used for coding for an application. Paragraph [0022] does not mention anything about processing an input object for the purpose of subsequently identifying code to be executed by a server. In addition, paragraph [0022] also fails to teach using a view list of at least one input element for processing a type of code identified by the input object as in the presently claimed invention, as *Christfort* only discloses the selection of input types to develop a client/server application. *Christfort* fails to teach using input type elements in the manner recited in the presently claimed invention. Furthermore, paragraph [0062] of *Christfort* merely discloses a host provider whose function is to install and maintain applications that are part of a service, and paragraph [0093] of *Christfort* merely discloses how a portal-to-go XML document or application program containing the code generates a portal-to-go XML document output, and how the portal-to-go XML document output is launched by the system. In contrast, claims 2 recites the processing of the generated code includes using a server

list of at least one server element for determining the one of the plurality of servers. *Christfort* does not teach and is not concerned with selecting the correct server to use for the type of code. *Christfort* discloses that a user may select a service manually, but does not support processing code to determine which server to use for the code type.

Claims 3, 14, 16, and 18 recite wherein the view list, server list and launcher list are extensible to accommodate additional respective elements, and claims 4 and 5 recite maintaining at least one of the view list, server list and launcher list. While the Examiner refers again to paragraphs [0022], [0062], and [0093] of *Christfort* as teaching these features, *Christfort* does not teach any extensibility feature of a view list, server list, and launcher list, nor does *Christfort* teach maintaining such lists.

Claims 6 and 7 recite that processing the input object comprises analyzing the input object to determine an input object element for processing the input object and processing the input object using the determined input object element and processing user input to determine the input object element and processing user input to determine the input object element. The Examiner refers to *Christfort* as teaching these features. The cited sections of *Christfort* teach identifying several types of input objects that can be used for coding for the application. However, the SDK provider in *Christfort* does not perform any processing of an input object for the purpose of subsequently identifying code to be executed by a server.

Claims 8, 9, and 15 recite that processing the generated code comprises analyzing a server element for enabling a deployable object and processing the deployable object using the determined server element and processing user input to determine the server element. The Examiner refers to *Christfort* as teaching these features. The cited sections of *Christfort* teach creating code that will work on any client device. In contrast, claims 8, 9, and 15 recite processing a deployment object that enables a server to enable a server side application to run on that server. *Christfort* does not teach any analysis of server side elements or teach wherein such analysis is used for enabling a deployable object.

Claims 10, 11, and 17 recite that identifying the one of the plurality of client applications comprises analyzing a launchable object to determine a client element for processing the launchable object and processing the launchable object using the determined client element. The Examiner refers to *Christfort* as teaching these features. The cited sections of *Christfort* only

teach creating server side applications that can run on any type of client. In contrast, claims 10, 11, and 17 recite identifying a client application by analyzing a launchable object to determine the client element to process the launchable object. *Christfort* does not teach any analysis of a launchable object to determine a client application for processing the launchable object.

Claim 35 recites performing a compatibility test of the input object, deployable object, and launchable object prior to processing the input object and display a result of the compatibility test to a user. The Examiner refers to *Uszok* as teaching these features. The cited sections of *Uszok* only teach verifying runtime properties are compatible with botMaster capabilities, such as the presence of required plugins or GUI version. However, no mention is made in *Uszok* of testing the compatibility of the input object, deployable object, and launchable object as recited in the presently claimed invention. *Uszok* does not teach such a compatibility test of objects prior to processing the input object, nor of displaying the test results to a user.

B. CONCLUSION

As shown above, the Examiner has failed to state valid rejections against any of the claims. Therefore, Appellants request that the Board of Patent Appeals and Interferences reverse the rejections. Additionally, Appellants request that the Board direct the Examiner to allow the claims.

Date: January 20, 2010

Respectfully submitted,

/Cathrine K. Kinslow/

Cathrine K. Kinslow
Reg. No. 51,886
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777

CLAIMS APPENDIX

The text of the claims involved in the appeal is as follows:

1. A method of executing code in a client-server environment by defining an extensible mechanism for executing said code on a server that, when deployed on a computer system, adapts an integrated development environment (IDE) to handle new code types, the method comprising:

identifying an input object on a client system, the input object identifying code for executing on one of a plurality of servers;

processing the input object to identify the code for executing on the one of the plurality of servers;

generating, in response to identifying the code for executing on the one of the plurality of servers, code for accessing the code for executing on the one of the plurality of servers;

processing the generated code to determine the one of the plurality of servers on which to execute the code, each code for executing on a server being able to execute on a particular server;

enabling the determined server to access the code for executing on the one of the plurality of servers;

identifying, based on the accessed code for executing on the one of the plurality of servers, one of a plurality of client applications for allowing the determined server to interact with the client system during processing of the code for executing on the one of the plurality of servers; and

processing the code for executing on the one of the plurality of servers.

2. The method of claim 1 wherein processing the input object to identify the code for executing on the one of the plurality of servers includes using a view list of at least one input element for processing a type of code identified by the input object, processing the generated code includes using a server list of at least one server element for determining the one of the plurality of servers, and identifying the one of the plurality of client applications includes using a launcher list of at least one client element for launching the one of the plurality of client applications.

3. The method of claim 2 wherein at least one of the view list, server list and launcher list is extensible to accommodate additional respective elements.

4. The method of claim 2 comprising maintaining at least one of the view list, server list and launcher list.

5. The method of claim 4 wherein the step of maintaining at least one of the view list, server list and launcher list comprises extending any of the view list, server list and launcher list.

6. The method of claim 1 wherein processing the input object comprises:

analyzing the input object to determine an input object element for processing the input object; and

processing the input object using the determined input object element.

7. The method of claim 6 including processing user input to determine the input object element.

8. The method of claim 1 wherein processing the generated code comprises:

analyzing a server element for enabling a deployable object; and

processing the deployable object using the determined server element.

9. The method of claim 8 including processing user input to determine the server element.

10. The method of claim 1 wherein identifying the one of the plurality of client applications comprises:

analyzing a launchable object to determine a client element for processing the launchable object; and

processing the launchable object using the determined client element.

11. The method of claim 10 including processing user input to determine the server element.

12. An extensible mechanism for executing server side code in a client-server environment that, when deployed on a computer system, adapts an integrated development environment (IDE) to handle new code types, the extensible mechanism comprising:

a view mechanism for processing an input object identifying code for executing on one of a plurality of servers and outputting a deployable object;

a server mechanism for processing the deployable object to determine the one of the plurality of servers for executing the code and to enable the deployable object to execute on the one of the plurality of servers, said second mechanism outputting a launchable object; and

a launcher mechanism for processing the launchable object to determine one of a plurality of client applications for launching the code on the one of the plurality of servers.

13. The extensible mechanism of claim 12 wherein said view mechanism comprises a view list of at least one input object element, each input object element processing a type of code identified by the input object for outputting the deployable object.

14. The extensible mechanism of claim 13 wherein said view list is extensible to accommodate additional respective elements.

15. The extensible mechanism of claim 12 wherein said server mechanism comprises a server list of at least one server element, each server element enabling the deployable object to execute on a particular server and processing the deployable object for outputting a launchable object.

16. The extensible mechanism of claim 15 wherein said server list is extensible to accommodate additional respective elements.

17. The extensible mechanism of claim 12 wherein said launcher mechanism comprises a launcher list of at least one client element, each client element enabling the launchable object to execute on a particular client for launching the code on the particular server.

18. The extensible mechanism of claim 17 wherein said launcher list is extensible to accommodate additional respective elements.

19. The extensible mechanism of claim 12 wherein said extensible mechanism is adapted to launch the one of the plurality of client applications determined in response to the launchable object for executing the code on the one of the plurality of servers.

20. The extensible mechanism of claim 12 wherein at least one of said view mechanism, server mechanism, and launcher mechanism is extensible whereby said view mechanism is extensible to accommodate a plurality of code types, said server mechanism is extensible to accommodate a plurality of servers and said launcher mechanism is extensible to accommodate a plurality of client applications.

21. The extensible mechanism of claim 12 wherein said view mechanism is adapted to analyze the input object to determine an input object element for processing the input object and process the input object using the determined input object element.

22. The extensible mechanism of claim 21 wherein said view mechanism is further adapted for processing user input to determine the input object element.

23. The extensible mechanism of claim 12 wherein said server mechanism is adapted to analyze the deployable object to determine a server element for processing the deployable object; and process the deployable object using the determined server element.

24. The extensible mechanism of claim 23 wherein said server mechanism is further adapted for processing user input to determine the server element.

25. The extensible mechanism of claim 21 wherein said launcher mechanism is adapted to analyze the launchable object to determine a client element for processing the launchable object;

and process the launchable object using the determined client element.

26. The extensible mechanism of claim 25 wherein said launcher mechanism is further adapted for processing user input to determine the server element.

27. The extensible mechanism of claim 12 wherein said extensible mechanism is adapted to be integrated into an integrated development environment.

28. A computer program product embodied in a computer readable medium having instructions that are to be executed by a processor to have a computer system perform a method in accordance with claim 1.

29. A computer readable media storing instructions to be executed by a processor of a computer system, said processor of the computer system executing an integrated development environment (IDE) for generating code for executing in a client-server environment, said instructions defining an extensible mechanism for executing said code on a server that, when deployed on said computer system, adapts said IDE for handling new code types to:

process an input object identifying code for executing on one of a plurality of servers, said processing using a view list of at least one input object element, each input object element processing a type of code identified by the input object to output a deployable object;

process the deployable object using a server list of at least one server element to determine the one of the plurality of servers for executing the code, each server element enabling the deployable object to execute on a particular server and outputting a launchable object; and

process the launchable object using a launcher list of at least one client element to determine a client for launching the code on the one of the plurality of servers.

30. The computer readable media of claim 29 wherein said IDE is further adapted for modifying at least one of the view list, server list and launcher list.

31. The computer readable media of claim 29 wherein said IDE is further adapted to launch the client determined in response to the launchable object to execute the code on the one of the plurality of servers.

32. A method of maintaining an extensible mechanism for executing server side code in a client-server environment by defining an extensible mechanism for executing said code on a server that, when deployed on a computer system, adapts and integrated development environment (IDE) to handle new code types, the method comprising:

maintaining at least one of:

a view list of at least one input object element, each input object element processing a type of code identified by the input object to output a deployable object;

a server list of at least one server element to determine one of a plurality of servers for executing the code, each server element enabling the deployable object to execute on a particular server and outputting a launchable object; and

a launcher list of at least one client element to determine one of a plurality of client applications for launching the code on the one of the plurality of servers.

33. The method of claim 32 wherein the step of maintaining comprises at least one of:

generating a respective element;

adding a respective element;

configuring a respective element; and

deleting a respective element from at least one of the view list, server list and launcher list.

35. The computer readable media of claim 29, further comprising:

perform a compatibility test of the input object, deployable object, and launchable object prior to processing the input object;

display a result of the compatibility test to a user.

EVIDENCE APPENDIX

This appeal brief presents no additional evidence.

RELATED PROCEEDINGS APPENDIX

This appeal has no related proceedings.